Supervised Learning: Discriminative Methods

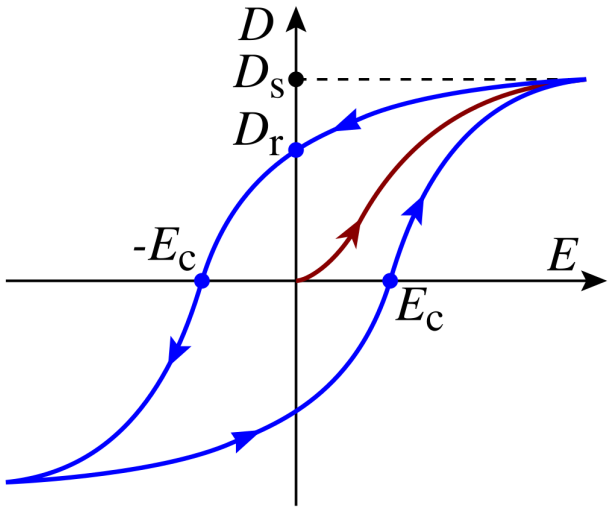
LSTM and CNN – Marco Laudato

ASSA School Series – Eindhoven, NL 05-11-2025

ASSA
AUTUMN SCHOOL SERIES IN ACOUSTICS

Recurrent Neural Network

RNN's general idea



- RNN can look at the data in context (temporal evolution).
- The information is weighted in time (from the more to the less recent).
- RNNs use a hidden state to condition on past information.

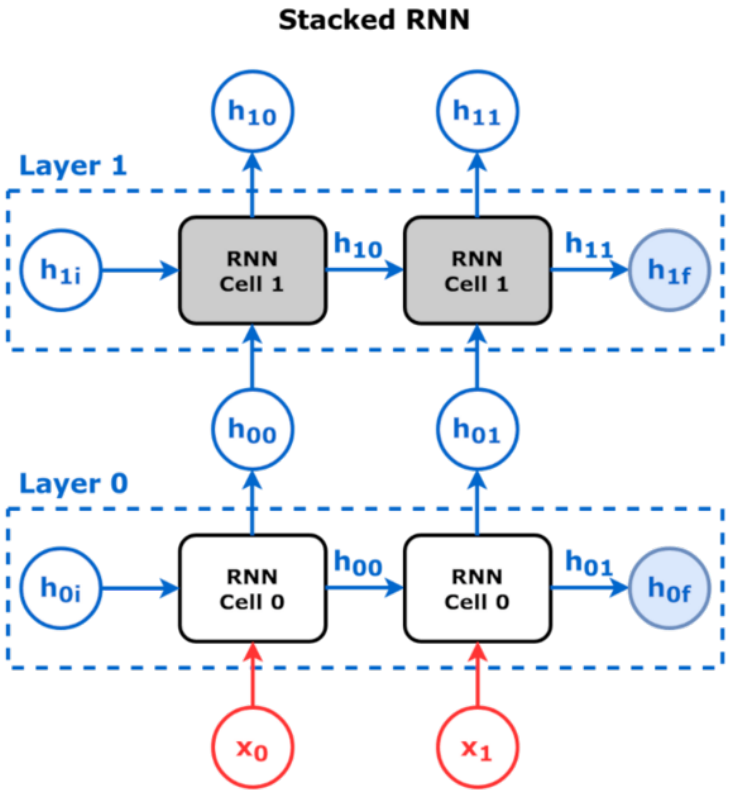
- Depending on the way the hidden state is defined (i.e., how past information is remembered) we have:
  - Gated Recurrent Units (GRU): simpler, works for smaller datasets.
  - Long-Short-Term Memory (LSTM): more complex "gating" mechanism. Revolutionised speech recognition, predictive/autocomplete text, …

- Typical applications: Time-series forecasting (weather, finance), speech recognition, language modelling, anomaly detections in streams, …

- Main limitation: vanishing gradients, numerical time differentiation.

RNN architecture

Consider the following 2 layers architecture:



- h_{ij} : general hidden state definition

$$h_{ij} = \tanh\left(W_x^{(i)} x_j + W_h^{(i)} h_{i,j-1} + b_i\right)$$

- Example (layer 0, timestep 1):

$$h_{01} = \tanh\left(W_x^{(0)} x_1 + W_h^{(0)} h_{00} + b_0\right)$$

- Example (layer 1, timestep 1):

$$h_{11} = \tanh\left(W_x^{(1)} h_{01} + W_h^{(1)} h_{10} + b_1\right)$$

- The outputs are computed using a linear definition:

$$y_j = W_y h_{1j} + c$$

- The hidden states carries the information from the previous time steps!

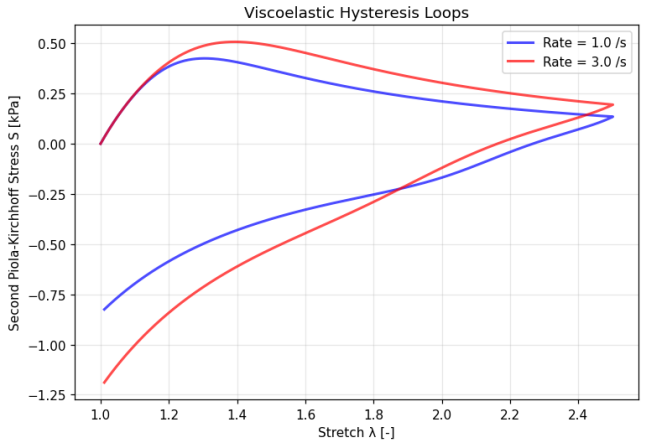
- Each layer uses its own unknown weight matrices W (shared across time steps).

RNN example

- Problem (same as hands-on): for a viscoelastic solid (Maxwell-type with power-law viscosity), the stress obeys a time-evolution law (ODE):

$$\dot{\sigma} = E\dot{\lambda} - \frac{\sigma}{\nu_0}\left|\frac{\sigma}{\sigma_0}\right|^{m-1}$$

- By increasing and then decreasing the stretch λ, the material exhibits hysteresis: the stress at time t depends on the loading history, not only on the current stretch.



- Goal: use a RNN to predict the stress at time t.

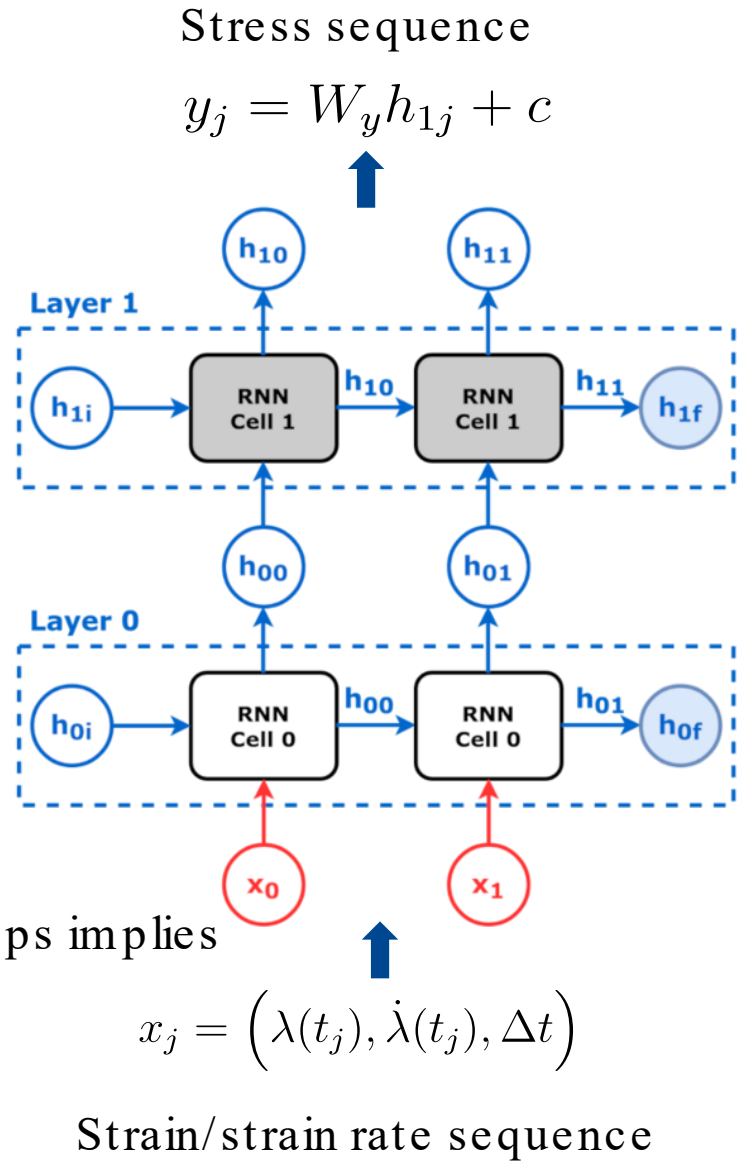
RNN training (example)

This is called a “sequence to sequence” problem:

- Input sequence: $x_j = \left(\lambda(t_j), \dot{\lambda}(t_j), \Delta t\right)$
- Output sequence: $y_j = \sigma(t_j)$
- The cost function: $\mathcal{L}_{\text{seq}} = \frac{1}{T}\sum_{j=1}^{T}\left(\sigma_j - \hat{\sigma}_j\right)^2$
- We will see that RNN can solve this problem!

--- Possible pitfall ---

- Vanishing or exploding gradient: weights shared across time steps implies that in backpropagation the gradient gets multiplied many times by the same Jacobian (if <1 the gradient vanishes!).



RNN vanishing/exploding gradients

- During back propagation (training) I must compute the Jacobian of the hidden state :

$$h_t = \phi(W_x x_t + W_h h_{t-1} + b) \quad \Longrightarrow \quad J_t = \frac{\partial h_t}{\partial h_{t-1}} = diag(\phi'(a_t))W_h$$

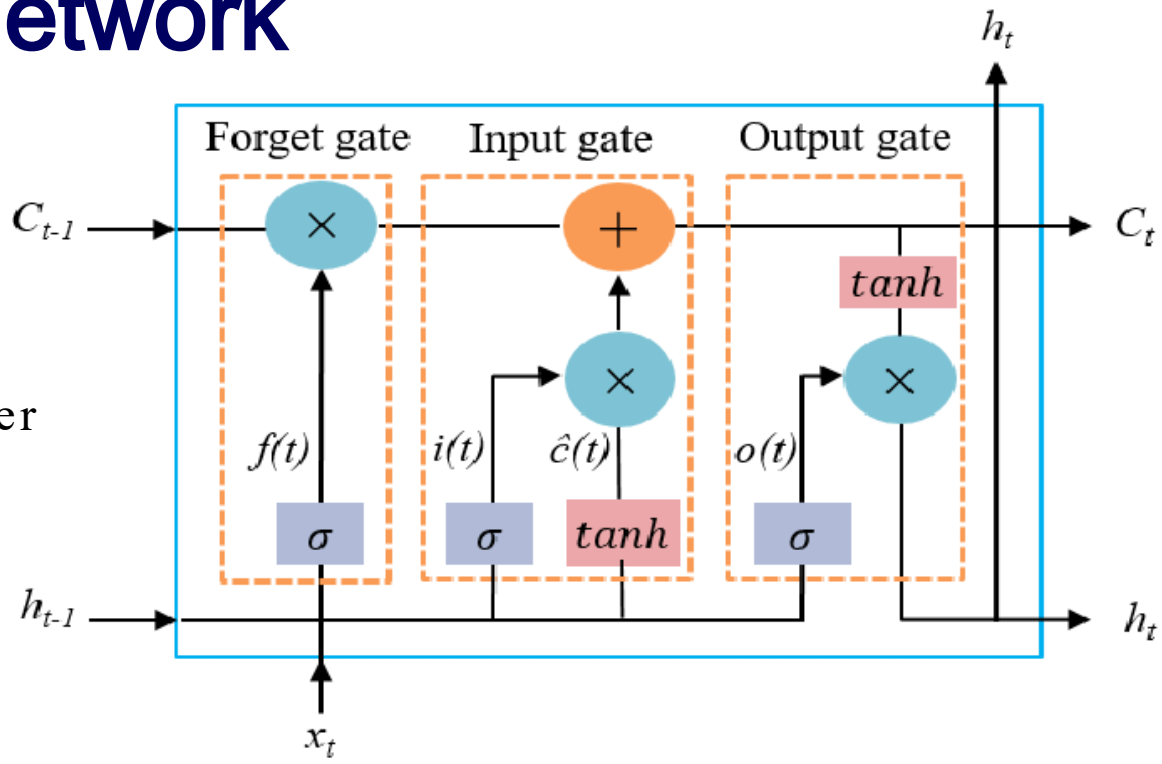
- Backpropagation multiplies many of these gradients:

$$J_t J_{t-1} J_{t-2} \cdots \sim \left(|W_h| \cdot |\phi'|\right)^k$$

- For different activation functions:
  - $\phi(\cdot) = \tanh(\cdot) \rightarrow \phi'(\cdot) = \mathrm{sech}^2(\cdot) \in (0, 1]$
  - $\phi(\cdot) = \mathrm{ReLU}(\cdot) \rightarrow \phi'(\cdot) \in \{0, 1\}$
  - $\phi(\cdot) = \dfrac{1}{1 + e^{-(\cdot)}} \rightarrow \phi'(\cdot) \leq 0.25$

Long-Short Term Memory network

- Every node of a LSTM produces *two* outputs:
  - **h**: hidden state (short-term memory)
  - **C**: cell state (long-term memory)
- The gates introduce new parameters.
- Intuition: gates tell us how much the network remember from the previous time-steps.



- Gates:

$$f = \sigma(W_f[x(t_j), h_{j-1}] + b_f) \in [0, 1]$$

$$i = \sigma\left(W_i[x(t_j), h_{j-1}] + b_i\right)$$

$$o = \sigma\left(W_o[x(t_j), h_{j-1}] + b_o\right)$$

Cell state update:

$$\hat{C}_j = \tanh\left(W_C[x(t_j), h_{j-1}] + b_C\right)$$

$$C_j = f \otimes C_{j-1} + i \otimes \hat{C}_j$$

hidden state update

$$h_j = o \otimes \tanh\left(C_j\right)$$

LSTM gradients

- The linear update of the cell state create a *linear highway* for the gradient:

$$\frac{\partial C_j}{\partial C_{j-1}} \sim \text{diag}(f)$$

- The information propagates through the depth of the network (many timestep → *long memory*).
- The hidden states gradient propagate as before (*short memory*):

$$\frac{\partial h_j}{\partial h_{j-1}} \sim o \cdot \text{sech}^2\left(C_j\right)$$

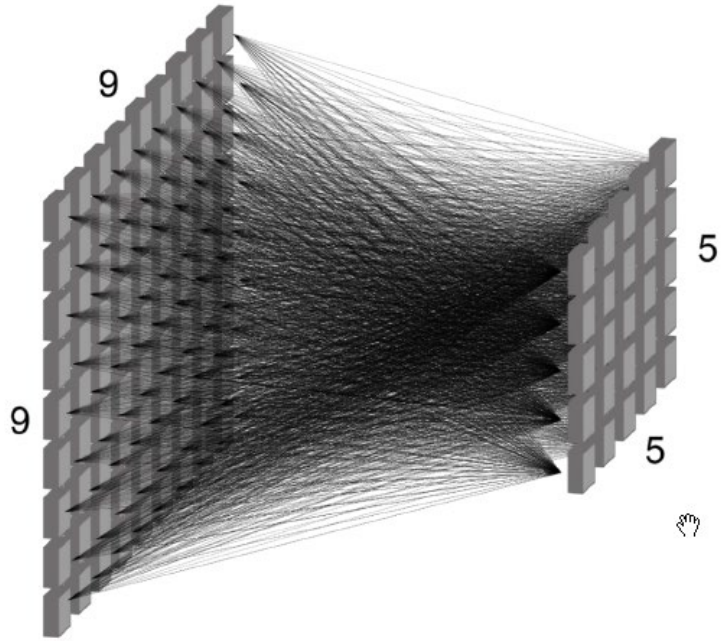
Convolutional Neural Network

Courtesy of Marcus Mäder

Introduction

Motivation – When the models explode

- Lets assume a fully connected layer for image processing
  - Input is a 9×9, output is 5×5, with 4 Byte per parameter
  - Memory: $9 \cdot 9 \cdot 5 \cdot 5 \cdot 4$ Byte results in 8kB
  - FullHD to FullHD: $1920 \cdot 1080 \cdot 3 \cdot 1920 \cdot 1080 \cdot 3 \cdot 4\text{Byte} = 154\text{TB}$
- Number of required training data increases with number of parameters

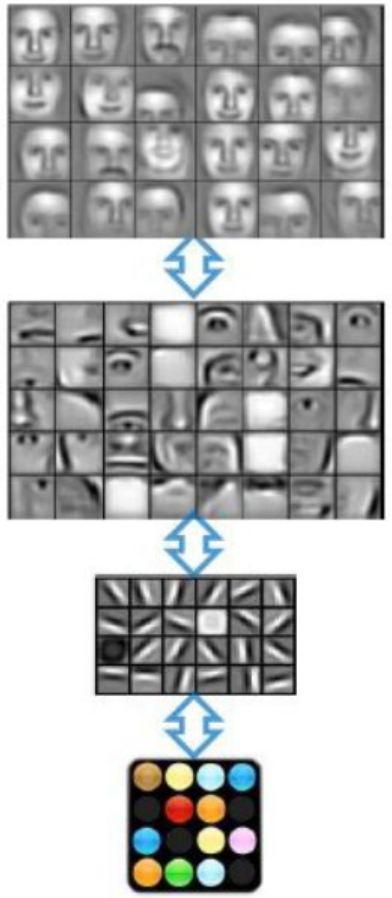


Source: Adopted from TUM lecture: Prof. Lienkamp “Artificial Intelligence in Automotive Technology”

Introduction

Convolutional Neural Networks

- Input
  - Pixels as vectors with corresponding information
- 1st layer (principle forms)
  - Edges, Corners, Circles
- 2nd layer (object parts)
  - Eye, Nose, Arm
- 3rd layer (objects)
  - Face, Body
- Goal: Automatically lean feature hierarchies with higher lever features formed by lower level features, without dependency of human-engineered features

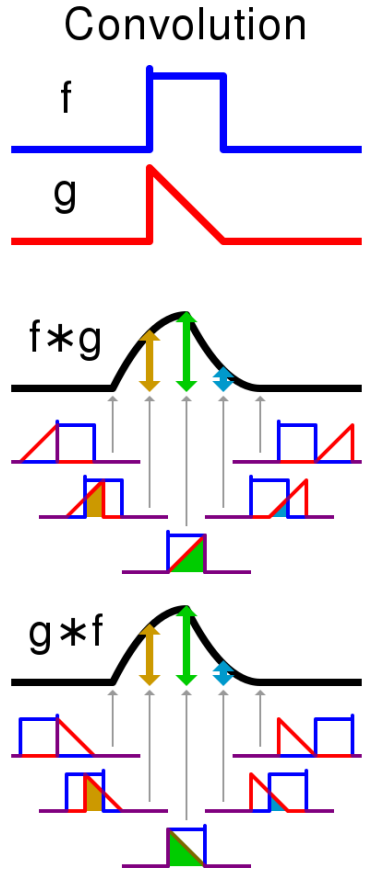


Source: https://deeplearningworkshopnips2010.files.wordpress.com/2010/09/nips10-workshop-tutorial-final.pdf

Convolution neural networks

Why convolutional?

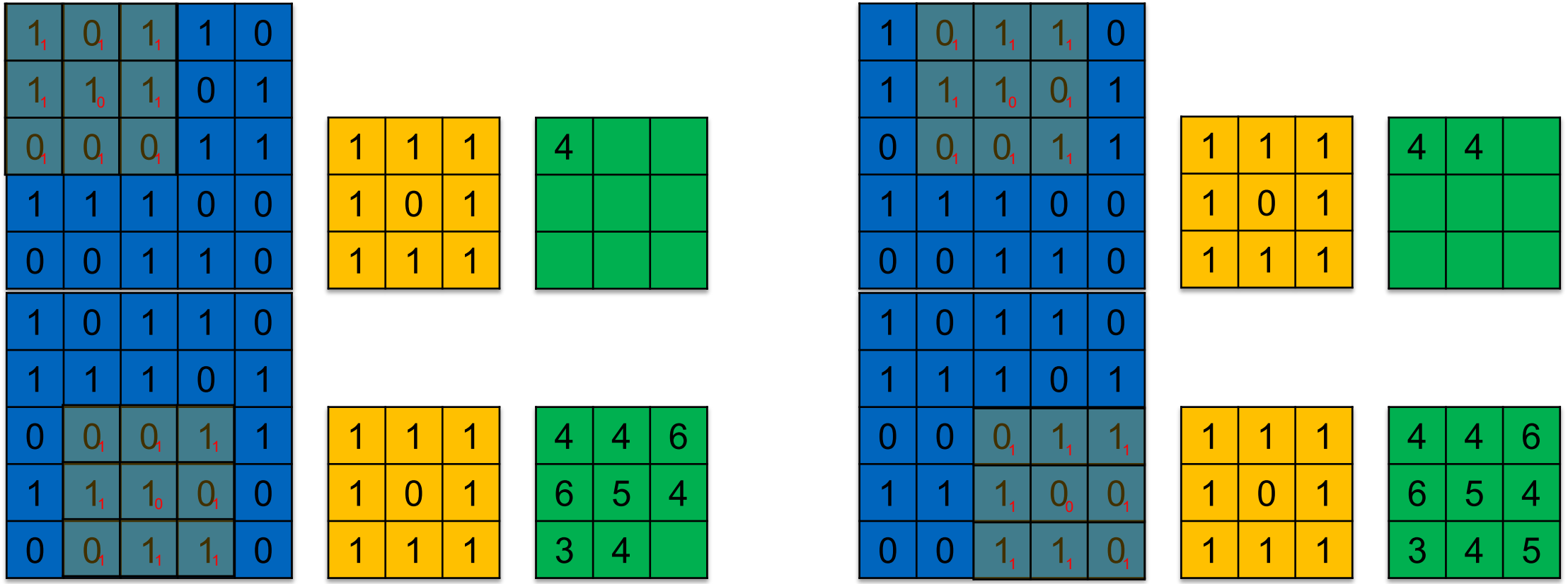
- Mathematical formulation $(f \star g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t-\tau) \mathrm{d}\tau$
- The convolution gives an idea on how function is influenced by another function
- Can be used to filter the original data
  - Filters can be understood as different features
  - Convolution results in spatial occurrence of features



Source: https://en.wikipedia.org/wiki/Convolution

Convolution neural networks

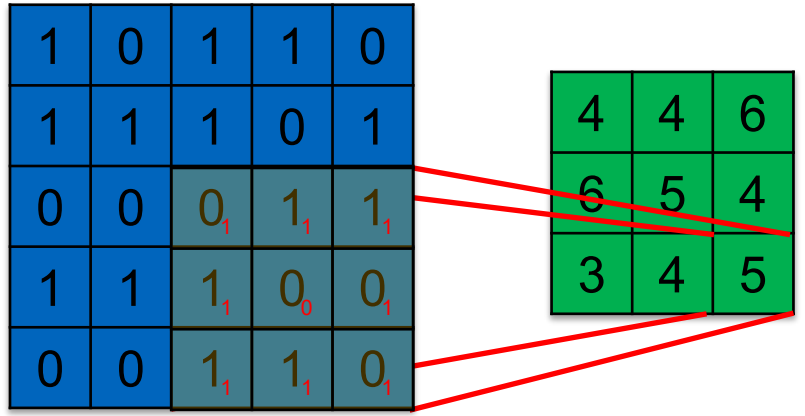
Application of convolution



Convolution neural networks

Features, Feature maps, and local receptive fields

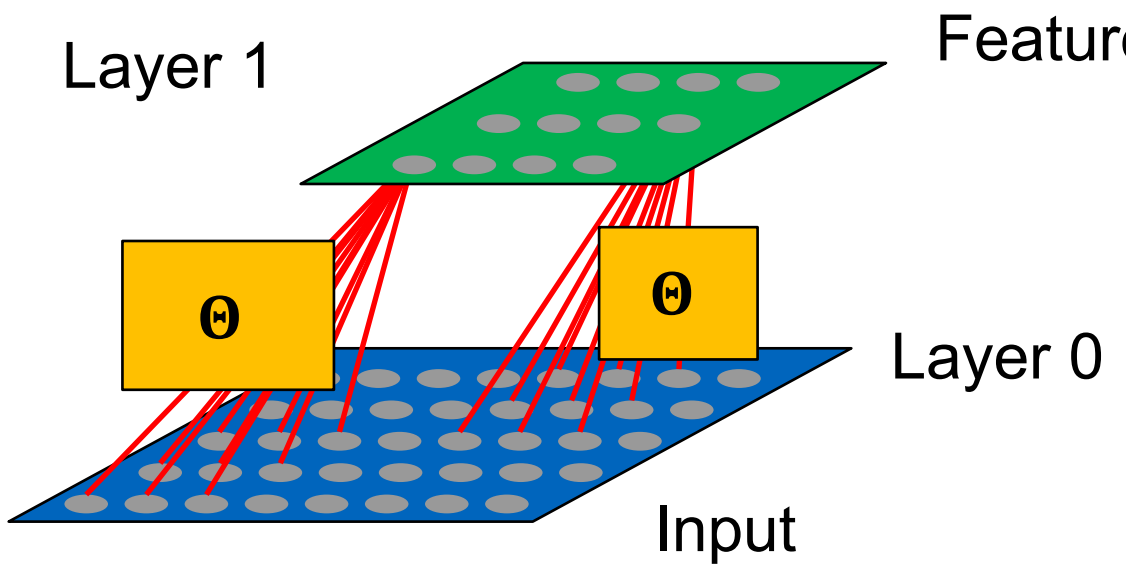
- Filter coefficients of feature f^k are the weights $\boldsymbol{\Theta}^k$ between one neuron in the hidden layer and its receptive field (connected neurons of the previous layer)
- For each feature f^k there exists a feature map
- The feature map contains as many entries as possible filter shifts
- All neurons of feature map have identical weights Θ^k
  - Shared weights
  - Low number of free parameters
- h_i^k is the i-th output of the k-th feature map
- $\boldsymbol{x}_i$ is the sub-region
- Output of corresponding hidden layer $h_i^k = g\left(\boldsymbol{\Theta}^k \boldsymbol{x}_i + \boldsymbol{b}^k\right)$



Convolution neural networks

Single feature map

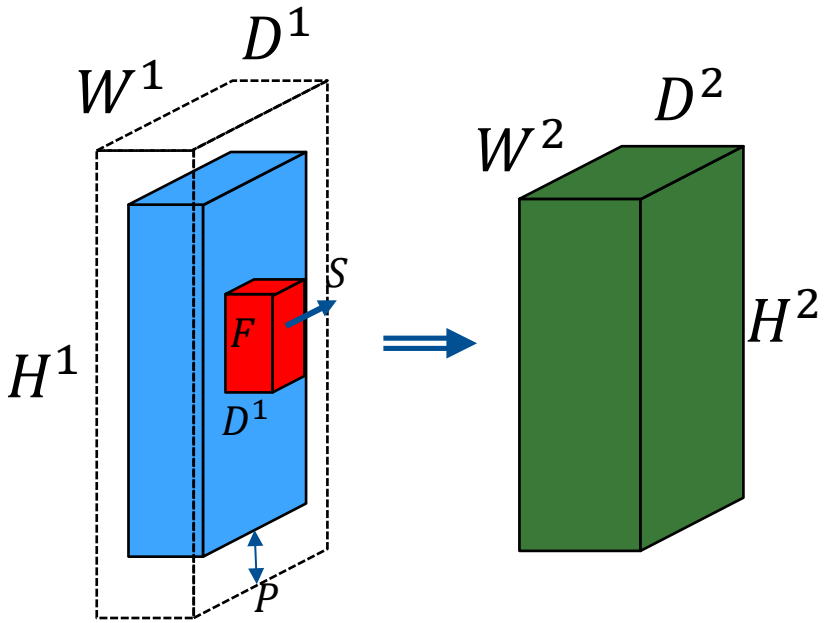
- All hidden neurons in the hidden layer share the same weight matrix



Convolution neural networks

Dimensions Formula

- Depending on the chosen convolution setting the dimension of hidden layer follows:
  - Number of Filters N
  - Filter size F
  - Stride S
  - Padding P
- New dimensions
  - $W^2 = \frac{W^1+2P-F}{S} + 1$
  - $H^2 = \frac{H^1+2P-F}{S} + 1$
  - $D^2 = N$



Keep dimensions:

$F = 3 \rightarrow P = 1$

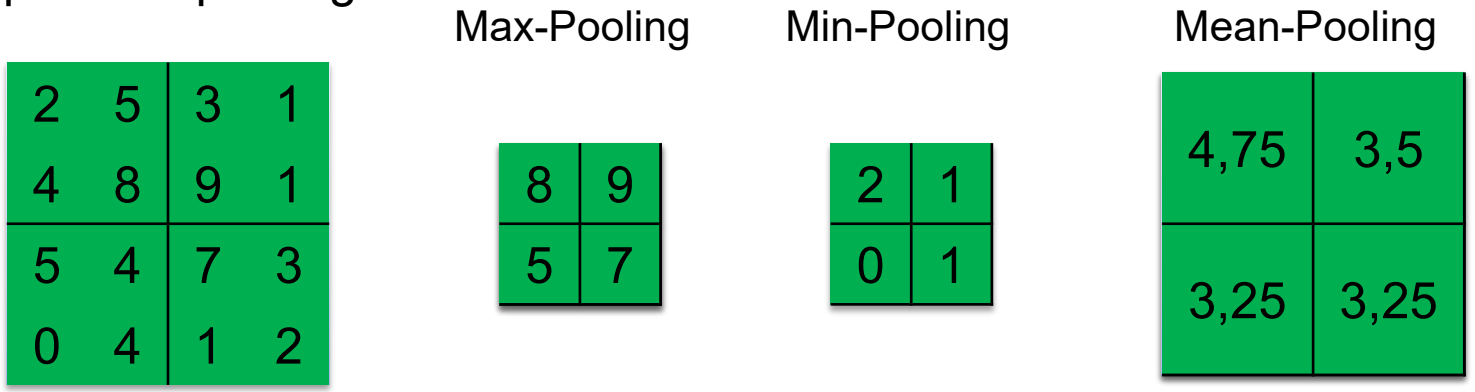
$F = 5 \rightarrow P = 2$

$F = 7 \rightarrow P = 3$

Convolution neural networks

Pooling

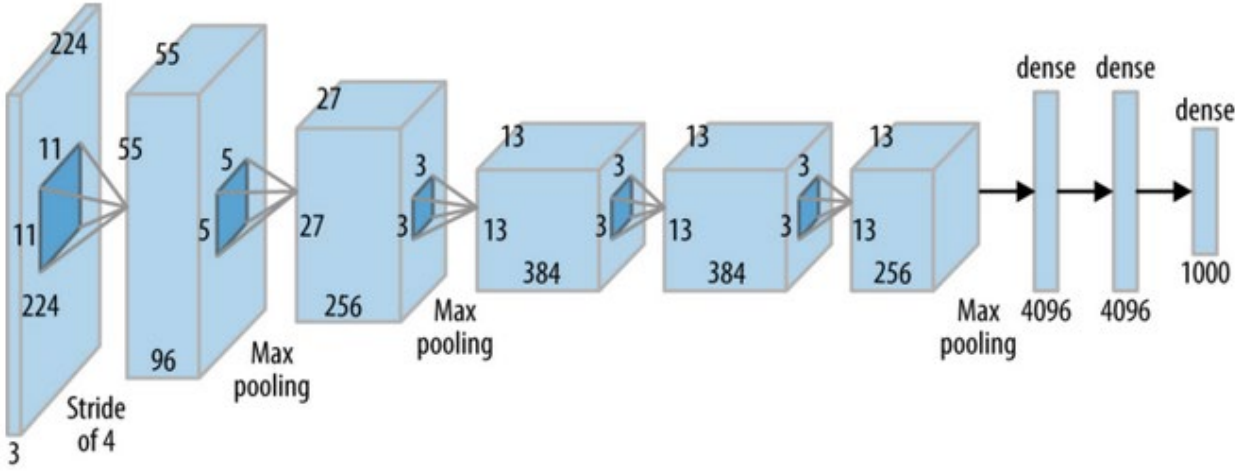
- Image classification requires some degree of translational/rotational invariance
- Reduces dimension, reduces overfitting, model becomes more tolerant
- Application
  - Partition feature map into a set of non-overlapping sub-regions
  - From these sub-region compute the pooling
    - Max-Pooling
    - Min-Pooling
    - Mean-Pooling



Convolution neural networks

Examples of complex CNNs

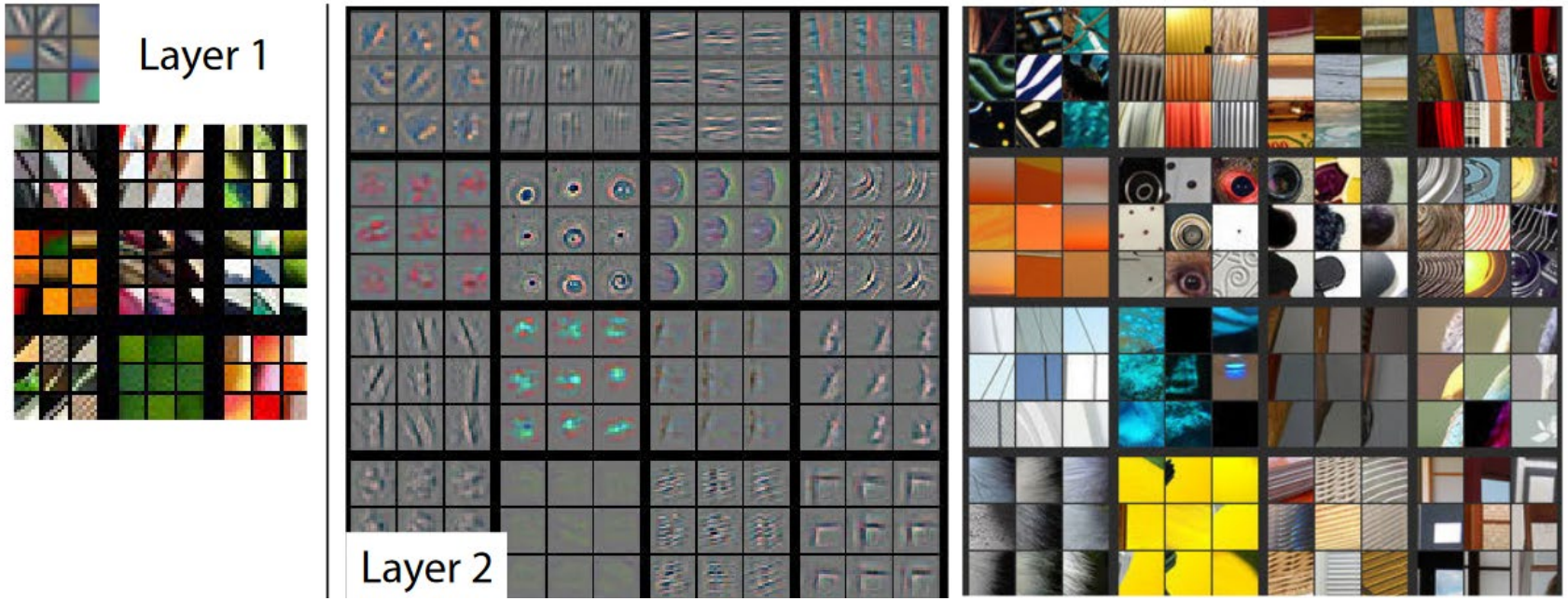
- AlexNet
  - Total of 62 million trainable variables



Source: https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

Convolution neural networks

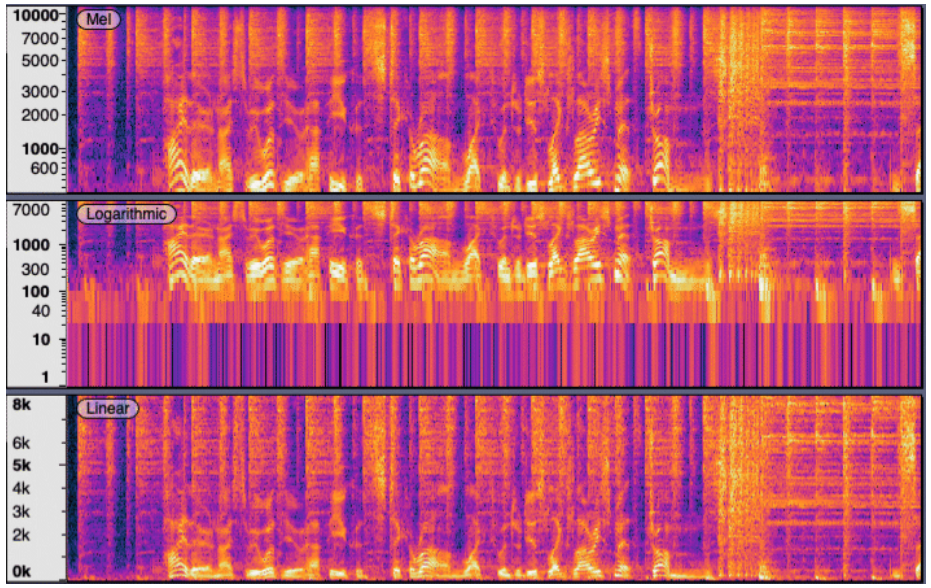
Feature representation



Source: https://www.matthewzeiler.com/mattzeiler/eccv2014.pdf

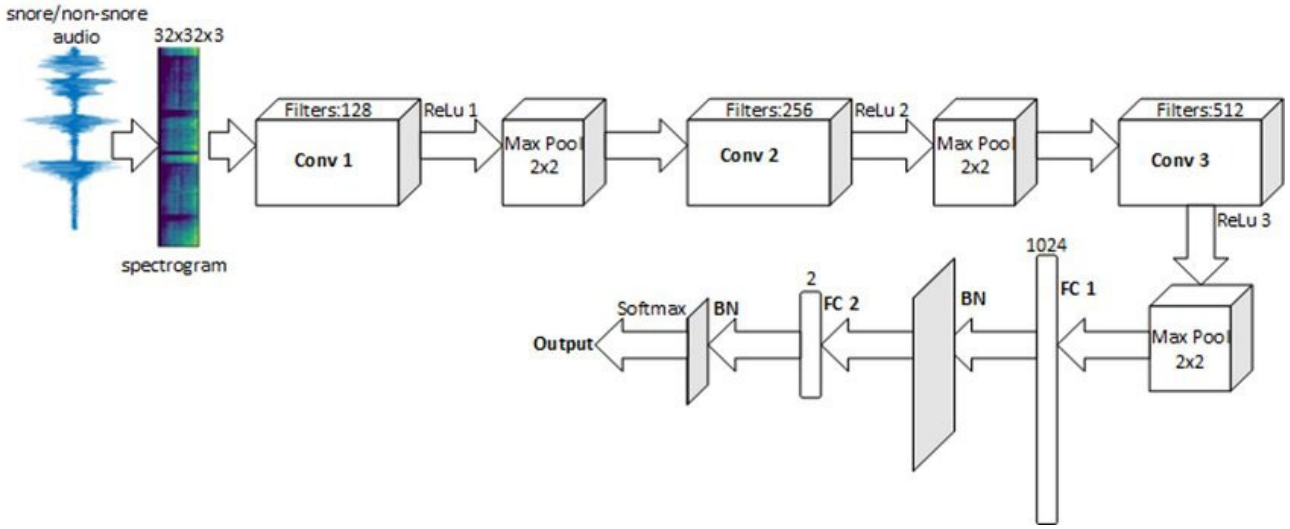
Sound as an image

- A spectrogram is a time-frequency visualisation of an acoustic signal: time on the horizontal axis, frequency on the vertical axis, and amplitude/intensity encoded by colour or brightness.
- By representing sound this way, we convert a 1-D waveform (function of time) into a 2 -D image (time ×frequency) suitable for spatial feature extraction using CNNs.



CNNs for spectrograms

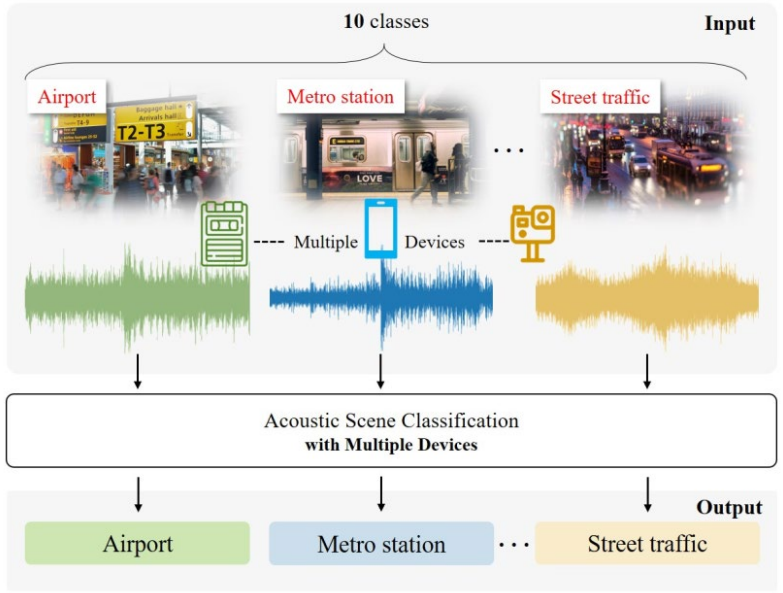
- The spectrogram image is fed into a 2-D convolutional neural network: convolutional layers scan across time-frequency axes, pooling layers reduce spatial dimensions, and fully connected layers perform classification or regression.
- Pre-processing is key (window length, hop size, Mel or linear scale, log amplitude)
- Interpretability: CNN filters may capture typical acoustic motifs (onsets, harmonics, broadband noise) and thus link back to physical acoustic phenomena.



Case study: acoustic scene classification

Acoustic Scene Classification:

- Applications:
  - auto-switch notification profiles,
  - hearing aids
  - smart buildings
  - automotive



Asthma Detection

- Asthmatic airways often produce characteristic acoustic patterns (wheezing, turbulent flow) which manifest as distinctive textures in spectrogram images.